Resumen

En la actualidad la electrónica está presente en cualquier automóvil. El bus CAN (*Controller Area Network*), patentado por Bosch en 1982, se ha convertido en un elemento indispensable para gestionar la gran cantidad de componentes electrónicos que se integran en un vehículo. La tarea de este bus es la de transmitir la información de manera fiable entre las diferentes ECUs (*Electronic Control Unit*). Por tanto, es factible conectarse a él y acceder al flujo de datos que envían los diferentes sensores, ECUs y actuadores del vehículo, y de esta manera registrar diferentes variables durante la conducción.

El objetivo de este trabajo es el desarrollo de un sistema de bajo coste y actualizable que permita monitorizar el comportamiento de un conductor y en particular detectar su grado de agresividad durante un ciclo de conducción. No obstante, debido a las restricciones de los fabricantes los cuales no revelan la codificación utilizada en las comunicaciones del bus CAN, esta caracterización del ciclo de conducción estará únicamente basada en el muestreo de la velocidad del vehículo, de la cual se puede obtener la aceleración.

Para ello se ha utilizado una conversor USB/CAN (USBtin) que realiza la función de interfaz CAN a USB (*Universal Serial Bus*) así como una tableta de bajo coste con sistema operativo Android, para la cual se ha desarrollado una aplicación específica programada en B4A (*Basic for Android*). Esta aplicación es capaz de registrar un ciclo de conducción, analizarlo una vez se ha completado, y proporcionar información en pantalla a un usuario no experto. A más bajo nivel la aplicación también ha de gestionar las tareas relacionadas con el intercambio de información a través del interfaz CAN a USB.

El funcionamiento de la aplicación ha sido validado a partir de pruebas con diferentes ciclos de conducción. Finalmente se incluye un estudio de las implicaciones medioambientales del proyecto así como un presupuesto económico.

Como conclusión del presente trabajo se puede decir que aunque el sistema que se ha desarrollado cumple con los requerimientos fijados al inicio del proyecto, su aplicación práctica es limitada debido a las barreras que impone la industria de la automoción al acceso a la información en los buses de comunicaciones.

Sumario

1. Glosario

API: Application Programming Interface

APK: Android application Package

B4A: Basic for Android

CAN: Controller Area Network

CCVS: Control Cruise Vehicle Speed

DLC: Data Length Code

ECU: Electronic Control Unit

GPS: Global Positioning System

GUI: Graphic User Interface

IDE: Integrated Development Environment

ISO: International Standards Organization

OBD: On Board Diagnostics

OTG: On the Go

PDU: Protocol Data Unit

PGN: Parameter Group Number

SAE: Society of Automotive Engineers

USB: Universal Serial Bus

2. Prefacio

2.1. Origen del proyecto

En los últimos años, y dentro del Departamento de Ingeniería Electrónica de la ETSEIB-UPC, se han venido realizando diversos trabajos y proyectos de fin de carrera dirigidos por el profesor Manuel Moreno relativos al uso del bus CAN en diferentes aplicaciones industriales y de automoción. A partir de esta experiencia previa, y considerando los recursos técnicos disponibles en la Universidad junto con el interés del autor por el tema, se planteó realizar el presente proyecto en el marco del Máster de Ingeniería de la Automoción.

En cierta manera este proyecto se puede considerar en línea con un proyecto previo del estudiante Marcel Garrido (Interfaz Gráfica de usuario usando una tableta Android y un adaptador CAN/USB) [1] dirigido por el mismo tutor, aunque revisado, ampliado y enfocado directamente al sector de la automoción. También se fundamenta en el uso de la tarjeta USBtin [2] desarrollada por Thomas Fischl, que permite el control de un bus CAN.

2.2. Motivación

En el pasado más reciente y en la actualidad, la trepidante evolución de los automóviles ha sido posible gracias a la introducción de la electrónica. Los sistemas electrónicos, que se podrían dividir en sensores, controladores y actuadores, permiten una gestión inteligente del comportamiento de un vehículo incrementando su seguridad, eficiencia y confort. Lógicamente estos sistemas electrónicos requieren de un canal que permita la comunicación entre ellos, siendo ésta la función del bus CAN, por el cual pasará la gran mayoría de la información que genera un automóvil y a la que se podrá acceder si se dispone de la base de datos adecuada para decodificarla e interpretarla.

Por otra parte, las tabletas son dispositivos extremadamente flexibles y configurables que en los últimos años han reducido su precio enormemente, lo que permite su utilización como sustituto de hardware diseñado específicamente, además de permitir su actualización de manera sencilla. Además de la tableta será necesario en este proyecto un interfaz CAN/USB que permita su conexión.

La concurrencia de ambos factores hace que actualmente ya sea posible monitorizar la información generada por un automóvil mediante un sistema de bajo coste, que es lo que se pretende en este proyecto.

3. Introducción

3.1. Objetivos del proyecto

El objetivo de este proyecto es el desarrollo de un sistema de bajo coste basado en el uso de una tableta Android que permita obtener información del ciclo de conducción de un automóvil. Esta tableta se conectará al bus CAN con la ayuda de un adaptador CAN/USB.

En particular y dentro de este proyecto se pretende como objetivo final que el sistema sea capaz de identificar situaciones en las cuales un vehículo pesado (camión) es conducido de manera brusca, y a partir de ellas evaluar el comportamiento de un conductor durante un ciclo de conducción. La caracterización del ciclo de conducción será mostrada en la pantalla de la tableta a un usuario no experto.

Por tanto, este proyecto a nivel operativo se centra en la programación de la tableta Android, por una parte para habilitarla para recibir información del bus CAN mediante el control de la tarjeta interfaz CAN/USB, por otra parte en lo referente al procesado y análisis de la información recibida, y finalmente todo lo que tiene que ver con el aspecto visual que se presenta al usuario final.

3.2. Alcance del proyecto

Tal y como se ha comentado en el prefacio, la información generada por un vehículo se transmite por el bus CAN, al cual nos podemos conectar de manera relativamente sencilla. No obstante, esto no implica directamente que podamos interpretar toda la información que viaja por él ya que está codificada. La codificación, guardada en bases de datos secretas y confidenciales, depende de cada fabricante y generalmente no está disponible (aunque la tendencia es a que en el futuro la legislación obligue cada vez más a compartir con terceros está información).

Este hecho implica una limitación en nuestro proyecto. Por este motivo sólo nos podemos basar en la información de la velocidad en camiones, cuya codificación sí que está regulada por el estándar J1939 [3]. Ciertamente sería posible obtener información de otros dispositivos (GPS, acelerómetros, etc.) pero se quiso desde el primer momento obtener información únicamente desde el bus CAN, utilizando únicamente los recursos integrados de serie en un automóvil.

Para alcanzar el objetivo del proyecto propuesto en el apartado anterior se definen las siguientes tareas y metas:

a) Documentación sobre el bus CAN: Estudio del funcionamiento del Bus CAN, en especial de los protocolos utilizados para el intercambio de información.

b) Caracterización de los ciclos conducción: Definición de las situaciones de conducción brusca a partir de los cambios en la velocidad del vehículo y del método para identificarlas. En este proyecto no se considerarán otras variables que indudablemente podrían ser útiles, puesto que como ya se ha comentado no se tiene acceso a la información de los fabricantes.

c) Implementación y desarrollo del software en la tableta Android: Esto incluye el control de la tarjeta interfaz CAN/USB, el análisis de los ciclos de conducción y la Interfaz Gráfica del Usuario (GUI).

d) Test y validación: Generación de diversos ciclos de conducción y comprobación del buen funcionamiento de la aplicación desarrollada.

e) Realización de un Presupuesto Económico y una evaluación del Impacto Ambiental.

3.3. Organización de este documento

Los contenidos del presente TFM se organizan de ahora en adelante de acuerdo a la siguiente estructura:

- En el capítulo 4 se realiza una introducción al funcionamiento del bus CAN, con especial atención a los aspectos más relevantes en relación al presente proyecto.
- El capítulo 5 está dedicado al funcionamiento de la interfaz CAN/USB USBtin.
- En el capítulo 6 se discute la manera de realizar una detección de la conducción agresiva a partir de la información disponible.
- En el capítulo 7 se explica la estructura del código fuente que se ha desarrollado, dando a conocer el funcionamiento de las rutinas programadas.
- En el capítulo 8 se exponen las funcionalidades de la aplicación, así como la metodología usada para su test y validación.
- Finalmente los capítulos 9, 10 y 11 se dedican a las consideraciones medioambientales, a la planificación temporal y al presupuesto del proyecto, respectivamente.

4. Fundamentos del bus CAN

En este capítulo se realiza una breve explicación del bus CAN, en especial en lo que atañe al desarrollo de este proyecto. Para una información más detallada se remite al lector a la amplia bibliografía existente sobre el tema [4], [5], [6].

4.1. ¿Qué es el bus CAN?

El bus CAN, desarrollado por Bosch en 1982, posibilita la comunicación entre los diferentes dispositivos electrónicos que existen en un vehículo. No obstante, dadas sus características se ha introducido en otros campos como las aplicaciones industriales y de control. Existen diferentes versiones del Bus CAN que atienden a diferentes especificaciones, en nuestro caso nos centraremos en la versión CAN 2.0 B [7]. En la actualidad el bus CAN se ha estandarizado a nivel internacional por la norma ISO 11898, y para aplicaciones en automoción por la SAE J22584 (turismos) y SAE J1939 (camiones y autobuses).

Adicionalmente el bus CAN permite realizar labores de diagnóstico OBD (*On Board Diagnostics*) a través de un conector situado en el vehículo que posibilita el acceso al bus CAN, tal y como se muestra en la Figura 1. Existen diferentes modelos de conectores estándar según el tipo de vehículo, así como cables adaptadores.

Figura 1. Acceso un conector CAN en un vehículo. Extraída de [8].

Es a través de este conector integrado en los vehículos donde conectaremos nuestro sistema de adquisición de datos.

4.2. Estructura del bus CAN

El bus CAN es un bus digital serie asíncrono donde todos los dispositivos conectados a él (nodos) comparten un mismo canal de comunicación (véase la Figura 2), lo que hace posible reducir el cableado en un vehículo -así como el peso y coste asociados-.

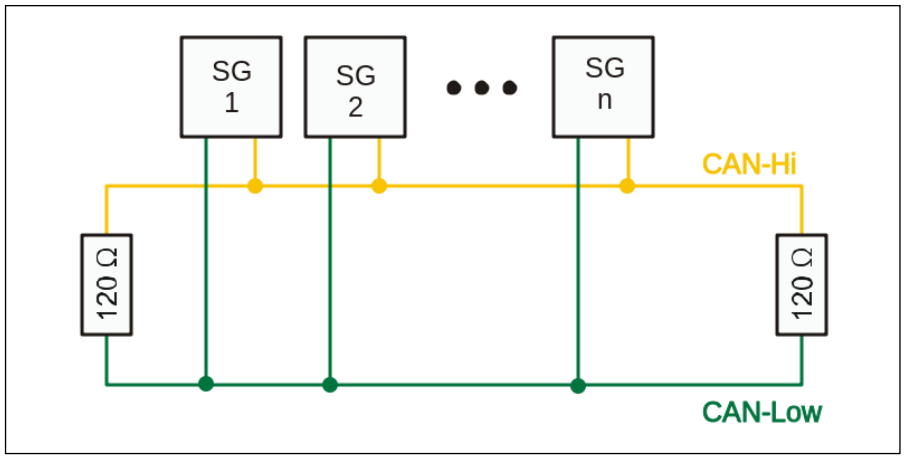

Figura 2. Esquema de un bus CAN con nodos conectados. Extraída de [4].

A nivel físico su estructura consiste en un par trenzado (y blindado en camiones) por el cual se envía una señal digital del tipo diferencial. Esto es importante porque al trabajar en un medio agresivo (calor, vibraciones, emisiones electromagnéticas, etc.) la prioridad principal -además de la robustez- es la fiabilidad, la cual se consigue en primer lugar reduciendo el ruido que pueda perturbar la señal eléctrica. Además, tal y como se muestra en la figura anterior, se debe terminar el cable con una resistencia terminadora para evitar reflexiones de la señal al final del bus.

La señal que se trasmite es binaria, obteniéndose de la diferencia entre *CAN_High* y *CAN_Low*.

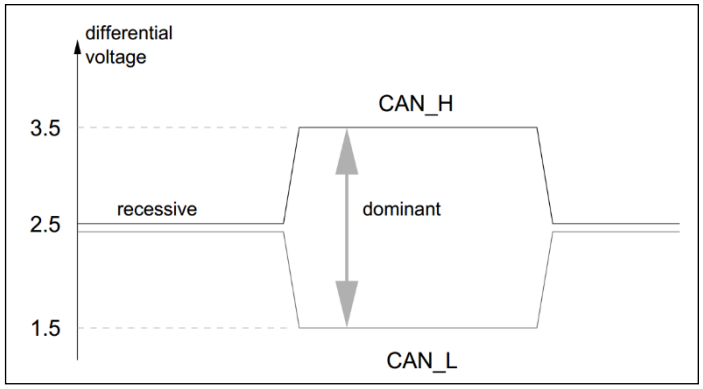

Figura 3. Niveles de tensión del bus CAN. Extraída de [5].

El bus CAN especifica dos estados definidos: estado dominante (“0” lógico) y estado recesivo (“1” lógico), tal y como se puede observar en la Figura 3. En estado recesivo, los dos cables del bus se encuentran al mismo nivel de tensión (tensión en modo común), mientras que en estado dominante hay una diferencia de tensión entre CAN_H (High) y CAN_L (Low) de al menos 1,5 V. La tensión en modo común puede estar entre -2 y 7 V.

Por otra parte, la velocidad de transmisión depende de la longitud del cable. En la práctica y en aplicaciones de automoción las velocidades del bus típicas se sitúan entre 125 Kbps y 500 Kbps.

Finalmente, puesto que la eliminación total del ruido es imposible, otro aspecto importante entonces será el protocolo utilizado, que ha de permitir detectar y corregir errores que se hayan producido durante la transmisión. Esto se denomina la capa de enlace de datos, que define como ha de trabajar la electrónica a bajo nivel así como la manera de gestionar la sincronización entre nodos, el arbitraje de las prioridades, etc. (acceso al medio). Para los fines de este proyecto los detalles no son relevantes, excepto en aquello referente a la estructura del mensaje, y se considerará cómo una caja negra ya que la tarjeta interfaz CAN/USB que utilizaremos se encarga de gestionar la mayoría de estas funciones.

4.3. Estructura de un mensaje CAN

De una manera simplificada la estructura (trama) de un mensaje CAN cuando se envían datos es la siguiente:

- Identificador (11 bits, extendido: 29 bits): Indica el tipo de mensaje que se está enviando. Además sirve para asignar la prioridad del mensaje. Cuando menor es el identificador, mayor es la prioridad del mensaje.

- DLC (4 bits): *Data Length Code*. Indica el número de bytes de datos que se van a transmitir en el mensaje. Puede valer entre 0 y 8.

- Datos (hasta 64 bits, 8 bytes como máximo): Mensaje que se envía, el valor de la velocidad del vehículo en nuestro caso.

Además de los campos anteriores también se incluyen en la estructura campos adicionales que no aportan información pero son necesarios para detectar el inicio y el fin del mensaje así como para detectar errores. En la Figura 4 se muestra el esquema de un mensaje CAN con todos los campos anteriores:

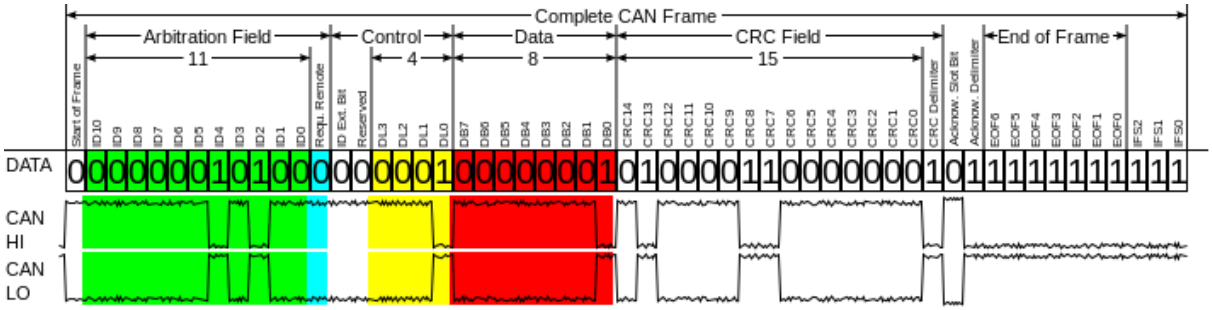

Figura 4. Estructura de un mensaje CAN. Se muestran los campos de Identificador en verde, DLC en amarillo y de Datos en rojo. Extraída de [4].

4.3.1. Otros tipos de mensajes

Por otra parte, además de un mensaje de datos también existen mensajes de petición de datos, mensajes de error y mensajes de sobrecarga de un nodo. No se entra en detalle en la explicación completa en este documento pero nuestro sistema ha de ser capaz de reconocerlos gracias al hardware utilizado.

4.4. Interpretación de un mensaje CAN

El siguiente nivel, que es el que nos interesa directamente, es la capa de aplicaciones, que define como se codifican los datos contenidos dentro de la estructura de un mensaje CAN descrita anteriormente. El problema surge si se desconocen cuáles son los identificadores que representan la información a la cual queremos acceder y tampoco se conoce como están codificados los datos. El estándar del bus CAN define la estructura del paquete donde viajan los datos, pero no especifica una manera estándar de codificarlos, luego cada fabricante puede establecer su propia forma de codificar los datos (llamado capa de protocolo de alto nivel). Algunos sí que se pueden conocer debido a que son necesarios para realizar diagnóstico, pero en principio la política del fabricante es la de la ocultación (en principio esto se justifica para no revelar secretos a la competencia pero otras veces el motivo es para evitar prácticas irregulares).

4.5. El protocolo de alto nivel SAE J1939

En el caso de vehículos pesados (camiones y buses) sí que existe un estándar definido en el estándar SAE J1939 [3] para codificar la información que nos permite interpretar la información recibida. Esto es así porque se debe asegurar la compatibilidad entre la cabeza tractora y el remolque, en el caso de que no correspondan al mismo fabricante. En este caso se especifica que el identificador tiene un formato conocido en formato extendido (29 bits), siendo la velocidad de transmisión de 250 Kbps.

El estándar define un formato para la parte de identificador de un mensaje CAN. El identificador siempre incluirá un *Parameter Group Number* (PGN), tal y como podemos observar en la Figura 5. Este PGN identifica de manera unívoca un grupo de parámetros. Estos grupos de parámetros (definidos en las especificaciones SAE J1939-71 [9]) combinan señales similares o asociadas.

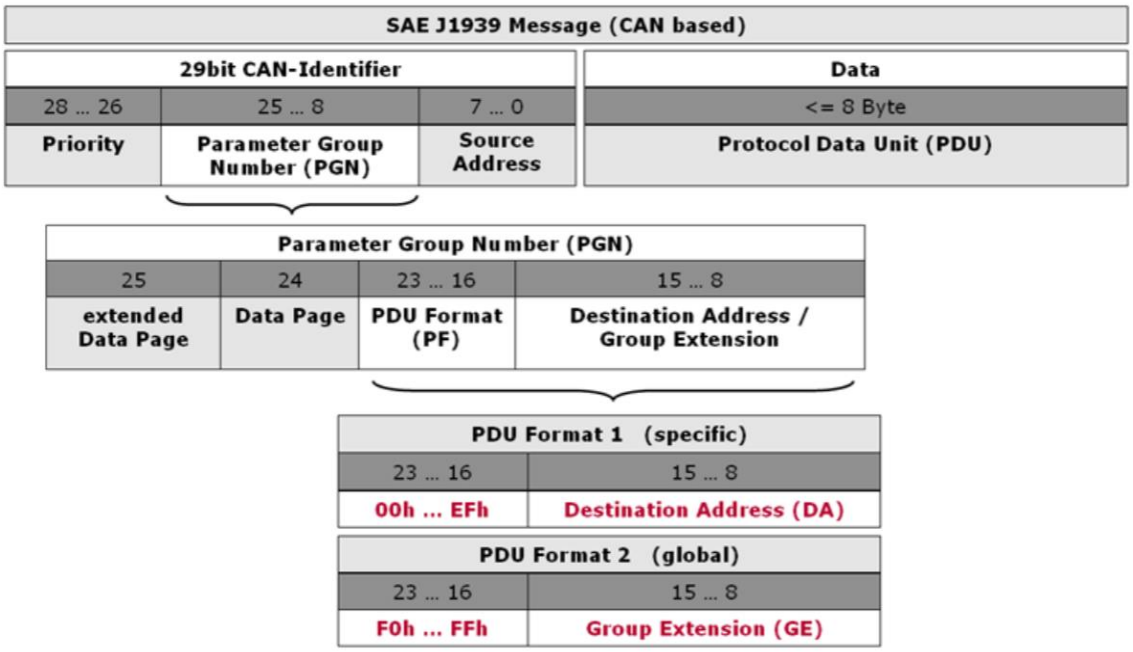

Figura 5. Formato de mensaje J1939 con el identificador de 29 bits. Extraída de [10].

A su vez, el PGN es una combinación de los bits de *extended Data Page* (reservado, siempre 0), el bit de *Data Page* y el PDU (*Protocol Data Unit*) Format i el PDU Specific (*Destination Address/Group Extension*).

Se pueden formar dos tipos de PGN:

a) Si PDU < 240 (peer-to-peer): PDU Specific contiene la dirección de destino (DA). Una dirección global (255) también puede utilizarse como dirección de destino.

b) Si PDU ≥ 240 (difusión): el formato PDU junto con la Extensión de Grupo (GE) en el PDU específico forma el campo del PGN del grupo de parámetros que se quieren transmitir .

Finalemente, los últimos 8 bits del identificador contienen la dirección del dispositivo que transmite el mensaje. La dirección es la etiqueta o handle, que se asigna para proporcionar una forma de acceder de forma exclusiva un dispositivo en la red. Para una red dada cada dirección debe ser única (254 disponibles). Esto significa que dos dispositivos diferentes (

ECU) no pueden utilizar la misma dirección.

4.5.1. Detalles de la codificación del parámetro velocidad en J1939

En nuestro caso si pretendemos monitorizar la velocidad del vehículo usaremos el parámetro *Wheel-Based Vehicle Speed,* que según el estándar [11] pertenece al grupo de parámetros *Cruise Control/Vehicle Speed* (CCVS) al que corresponde el PGN 65265 ó FEF1 en hexadecimal.

Es un PGN global

- Nombre: *Cruise Control/Vehicle Speed* (CCVS)
- Periodo de transmisión: 100 ms
- Longitud de datos (DLC): 8 bytes
- Extended Data Page: 0
- Data Page: 0
- PDU Format: FE
- PDU Specific: F1
- Prioridad por defecto: 6
- PGN: 65265 (18FEF100 h)

Si construimos el identificador de 29 bits a partir de la información anterior se obtendrá lo que se muestra en la Tabla 1:

	Prioridad			R	D	PDU Format								PDU Specific								Source Address (SA)							
Hex	1	8				F				E				F				1				0				0			
Dec						65265																0							
Bin	1	1	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0
bit	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Tabla 1. Identificador utilizado para el grupo de parámetros CCVS

Por tanto, el identificador tendrá el código 0x018FEF100 (419361024 en decimal), que es el que utilizaremos para comprobar que el mensaje CAN que recibamos corresponde a los datos que queremos registrar, es decir, la velocidad.

Por otra parte, según las especificaciones del estándar, la información de la velocidad se encuentra codificada en los bytes 2 y 3 del campo de datos, como se puede observar en la

descripción del campo de datos en la Tabla 2:

Byte 1	Two Speed Axle Switch (1) Parking Brake Switch (3) Cruise Control Pause Switch (5) Park Brake Release Inhibit Request (7)
Byte 2	Wheel-Based Vehicle Speed
Byte 3	Wheel-Based Vehicle Speed
Byte 4	Cruise Control Active (1) Cruise Control Enable Switch (3) Brake Switch (5) Clutch Switch (7)
Byte 5	Cruise Control Set Switch (1) Cruise Control Coast (Decelerate) Switch (3) Cruise Control Resume Switch (5) Cruise Control Accelerate Switch (7)
Byte 6	Cruise Control Set Speed
Byte 7	PTO Governor State (1) Cruise Control States (6)
Byte 8	Engine Idle Increment Switch (1) Engine Idle Decrement Switch (3) Engine Test Mode Switch (5) Engine Shutdown Override Switch (7)

Tabla 2. Parámetros incluidos en el campo de datos del grupo de parámetros CCVS

El estándar también fija la información necesaria para decodificar estos dos bytes:

- Resolución: 1/256 km/h por bit, 0 offset
- Rango de datos: De 0 a 250,996 km/h

Esto significa que el valor (en base 10) de los dos bytes se ha de dividir entre 256 para obtener la velocidad en km/h. En nuestro caso, como veremos más adelante, recibiremos la información en una cadena de bytes y calcularemos la velocidad a partir de la siguiente expresión:

$$velocidad=(MSB\cdot 256+LSB)/256 \qquad [Ecuación\ 1]$$

Donde MSB es el byte más significativo y LSB es el menos significativo.

5. Interfaz CAN a USB

En este proyecto se pretende conectar una tableta con un puerto USB (Universal Serial Bus) a un bus CAN. Por tanto es imprescindible disponer de un adaptador o interfaz que capte la señal transmitida por el par de cables trenzados como si fuese un nodo de la red y la convierta en una información legible por un puerto USB de un ordenador o tableta.

Existen bastantes interfaces CAN/USB comerciales [12]. Por ejemplo, Kvaser es una empresa que fabrica este tipo de dispositivos, que además requieren de un software (por ejemplo CANalyzer de Vector Informatik) y un PC para interpretar los datos. No obstante, su precio es elevado, y además el software no funciona con una tableta Android. Por tanto, en nuestro caso utilizaremos el interfaz de bajo coste USBtin y desarrollaremos nuestro propio software.

5.1. Tarjeta USBtin

USBtin [2] es una tarjeta de bajo coste diseñada por Thomas Fischl a partir de un proyecto con filosofía *Open Source*, lo cual tiene la ventaja de que hay mucha información disponible. Tiene un precio reducido (alrededor de 40 euros) y cubre todas nuestras necesidades.

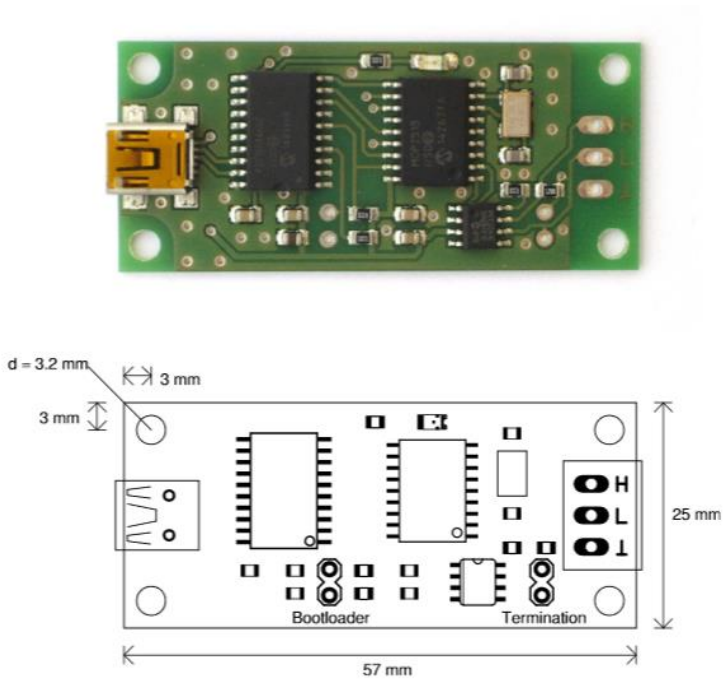

Figura 6. Fotografía y Esquema de la Tarjeta USBtin. Extraída de [2].

Además, existe software en la red desarrollado para esta tarjeta que permite conectarla a un

PC (Windows/Linux) aunque no a una tableta Android. Por otra parte, hay que tener en cuenta que no es un producto para aplicaciones comerciales, es decir, no cumple con las regulaciones pertinentes (CE, etc.).

A nivel físico, tal como se puede observar en la Figura 6, dispone de tres pines correspondientes al bus CAN (CAN_High, CAN_Low y Masa) en un lado y un conector USB hembra en el otro.

Esta tarjeta se controla mediante el puerto USB, al cual podemos enviar mensajes ASCII para enviar un mensaje CAN, o para recibirlos. También se pueden configurar diversos aspectos de su funcionamiento. Por otra parte, es importante mencionar que el puerto USB de la tableta debe ser del tipo OTG (*On The Go*), esto significa que la tableta puede comportarse como esclavo cuando se establece una conexión con la tarjeta interfaz.

Se omite aquí el detalle de control de esta tarjeta, el cual se explicará en el capítulo de desarrollo del software.

5.1.1. Timestamping

Un aspecto a resaltar es la posibilidad de activar el *timestamping*, esto es añadir después del mensaje CAN el tiempo correspondiente al instante en que el mensaje CAN fue recibido por la tarjeta, que podría ser significativamente diferente al tiempo en que nuestra tableta o PC reciba el mensaje por el puerto USB.

5.1.2. Máscaras y filtros

En un bus CAN todos los nodos tienen permiso para leer el tráfico de toda la red, pero cuando reciben un mensaje, estos nodos pueden realizar un test de aceptación para determinar si lo procesan o no (filtrado de mensajes).

USBtin permite configurarse para establecer máscaras que realicen el filtrado de mensajes de forma automática por hardware. La máscara se utiliza para indicar cuáles de los bits del identificador se utilizarán para realizar el proceso de filtrado.

Este filtrado lo realiza el propio hardware de la electrónica de la tarjeta y permite aligerar la carga de trabajo aguas abajo del filtro, al no procesarse los mensajes que no se van a necesitar.

Lamentablemente USBtin solo acepta identificadores de 11 bits, mientras que nosotros necesitamos 29 bits (formato extendido), por lo que no se puede hacer uso de esta característica de la tarjeta.

5.2. Kvaser

Kvaser (véase Figura 7) es otro interfaz CAN/USB que permite la comunicación entre el puerto USB y un bus CAN. Se puede conectar a un PC y permite establecer comunicación con el bus CAN (mediante un conector tipo DSUB 9). El precio de este dispositivo está en torno de los 300 euros

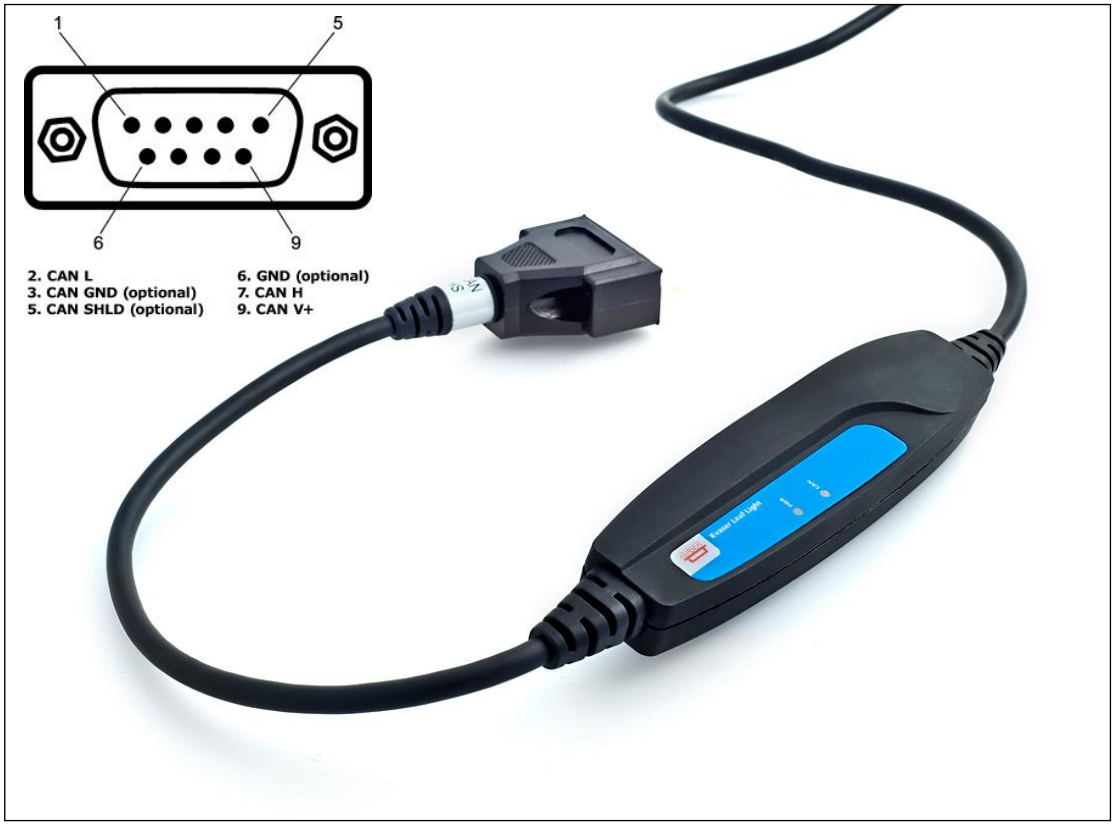

Figura 7. Kvaser Leaf Light HS. Fuente [13].

Puesto que se encuentra disponible en el Departamento de Ingeniería Electrónica, se utilizará para testear nuestro sistema, emulando el comportamiento del nodo (ECU del camión) que envía la velocidad del vehículo al bus CAN. Para ello, se utilizará un código en Python basado en el paquete CANLIB, que contiene las librerías y drivers necesarios para trabajar en entorno Windows.

6. Caracterización de los ciclos de conducción

En este capítulo se presenta el método utilizado para caracterizar los ciclos de conducción, que posteriormente se implementará en el software desarrollado.

6.1. Caracterización de un ciclo de conducción

La caracterización de un ciclo de conducción es el proceso por el cual se intenta responder a la pregunta de cómo ha sido conducido un vehículo. Esto puede tener diferentes respuestas si nuestra atención se centra en el consumo, la seguridad, el confort, etc. Pero aun así, un parámetro importante a medir sería la agresividad, la cual podemos imaginar que es función de los cambios en el vector velocidad del vehículo, o sea de su aceleración. Es evidente que una medida definida de tal manera interesa que sea baja en cualquier aspecto de una conducción, ya que comporta menos consumo de energía y menor riesgo de accidentes.

La caracterización de los ciclos de conducción es de interés en diferentes aplicaciones: control de emisiones [14], gestión de la energía de vehículos eléctricos o híbridos [15] [16], bonificación de las primas de seguros a los conductores, mejora del confort de los autobuses, monitorización del trato a vehículos de alquiler, etc.

No obstante, la caracterización de los ciclos de conducción puede ser una tarea compleja, en especial si se quiere separar la agresividad del conductor de la agresividad de la conducción (según la terminología propuesta aquí por el autor del presente proyecto). La agresividad de la conducción se podría definir de una manera absoluta mientras que la agresividad del conductor sería relativa a la mostrada por otros conductores en la misma situación.

La agresividad de la conducción viene determinada por la agresividad del conductor sometido a unas condiciones del flujo de tráfico. Es de esperar que si el conductor no tiene condicionantes externos (otros automóviles, señales de tráfico, etc.) la conducción sea más suave que si se imponen restricciones a su movimiento. Por tanto, un mismo trayecto puede tener una agresividad de la conducción diferente para un mismo conductor si las condiciones externas son diferentes. O a la inversa, dos conductores con diferente agresividad intrínseca pueden dar como resultado una agresividad de la conducción igual si las condiciones del trayecto son diferentes.

Lo ideal sería poder conocer la agresividad del conductor, que tiene mayor interés puesto que el objetivo sería advertirle de su mala conducción, pero para ello se tendría que extraer el efecto de los condicionantes externos. Esto solo sería posible comparando diferentes

conductores sometidos a los mismos condicionantes para un mismo trayecto. Después se podría ajustar un modelo (clasificador) válido en cualquier condición [17]. Este es un tema de interés en la literatura científica que es abordado por diferentes estudios [18], pero imposible de realizar con los medios disponibles, saliéndose del alcance de este TFM.

En nuestro caso nos fijamos un objetivo más asequible, que es la determinación de la agresividad de la conducción, considerada como aquella que surge de la interacción del conductor con las restricciones del flujo impuestas. Además, tal y como se ha comentado en el capítulo precedente, ha de estar basada únicamente en los datos de la velocidad (longitudinal) del vehículo.

Esto en principio establece una limitación en la aplicación de este método, pero no será relevante en nuestro caso de estudio, en el que se puede considerar que el trayecto siempre es el mismo y el tráfico no impone restricciones importantes a la circulación.

6.2. Método propuesto

El método que se propone se basa en el cálculo de la aceleración del vehículo obtenido a partir del muestreo de la velocidad.

En primer lugar se ha de fijar un umbral para el cual una aceleración se considera excesiva, ya que pequeños cambios en la aceleración son inherentes a la conducción. Para ello se ha de determinar un valor de referencia que se considere razonable.

El valor propuesto se determina a partir del estudio de los siguientes casos [19] que se presentan en la Tabla 3:

Caso	**Aceleración (g=9,8m/s2)**
Despegue avión comercial	0,5 g
Aceleración metro	0,14 g
Aceleración brusca coche	0,2 g
Frenada brusca coche	0,7 g

Tabla 3. Valores de la aceleración en diferentes casos. Fuente [19]

Como puede apreciarse en la Tabla 3, un valor de 0,14 g ya es apreciable, y es una referencia para los medios de transporte donde los pasajeros pueden viajar de pie. Valores

superiores a éste se pueden considerar muy agresivos.

Por otra parte, hay que comentar que en términos energéticos es más costoso acelerar cuando la velocidad ya es elevada que cuando es pequeña, ya que la energía cinética depende de la velocidad al cuadrado. Podríamos definir la agresividad en términos de variación de energía cinética si nos enfocáramos a determinar una agresividad para minimizar el consumo -por ejemplo para aplicaciones en coches eléctricos-. Puesto que no es nuestro objetivo, no lo haremos de esta forma.

Por tanto, se considerará el valor umbral 0,15g como aquel en el cual hay una aceleración agresiva. Valores menores se considerarán como "ruido" provocado por los esfuerzos del conductor para adaptarse al tráfico.

A partir de este valor se detectará el número de episodios donde se produce y se caracterizará en un primer momento la agresividad a partir del número de veces que se supera dicho umbral.

6.2.1. Indicador de agresividad

Puesto que no se dispone de las características del vehículo (masa, potencia, coeficiente aerodinámico) no es posible definir una agresividad específica para el vehículo. No obstante *Ford Scientific Research Laboratory* define en [20] la agresividad a partir de la siguiente expresión [Ec. 2]:

$$agresividad = \sqrt{\frac{1}{N}\sum(2 \cdot v \cdot a)^2} \qquad \textit{Ecuación 2}$$

Donde v y a son la velocidad y la aceleración de cada muestra respectivamente. En este trabajo se adoptará esta definición de la agresividad para caracterizar el ciclo de conducción junto con otras variables estadísticas.

.

7. Implementación del software en la tableta

En este capítulo se describe el proceso seguido en el desarrollo del software en una tableta Android. El código completo y comentado se encuentra disponible al final del documento en forma de anexo.

7.1. Elección del entorno de desarrollo

En base a experiencias anteriores se ha optado por el entorno de desarrollo que proporciona B4A (*Basic for Android*), que cuenta como ventaja respecto a otras alternativas con una curva de aprendizaje rápida, lo cual permite unos tiempos de desarrollo ajustados a la duración del proyecto. También cuenta con una comunidad bastante activa de usuarios, donde es posible formular preguntas, encontrar códigos de ejemplo, etc.

B4A cuenta con un entorno de desarrollo (IDE) que permite diseñar la pantalla que verá el usuario de la aplicación de una forma visual, enlazando luego los objetos creados al código del programa (véase Figura 8). Para cualquier detalle referente a las características o programación en B4A se remite al lector a su página web [21].

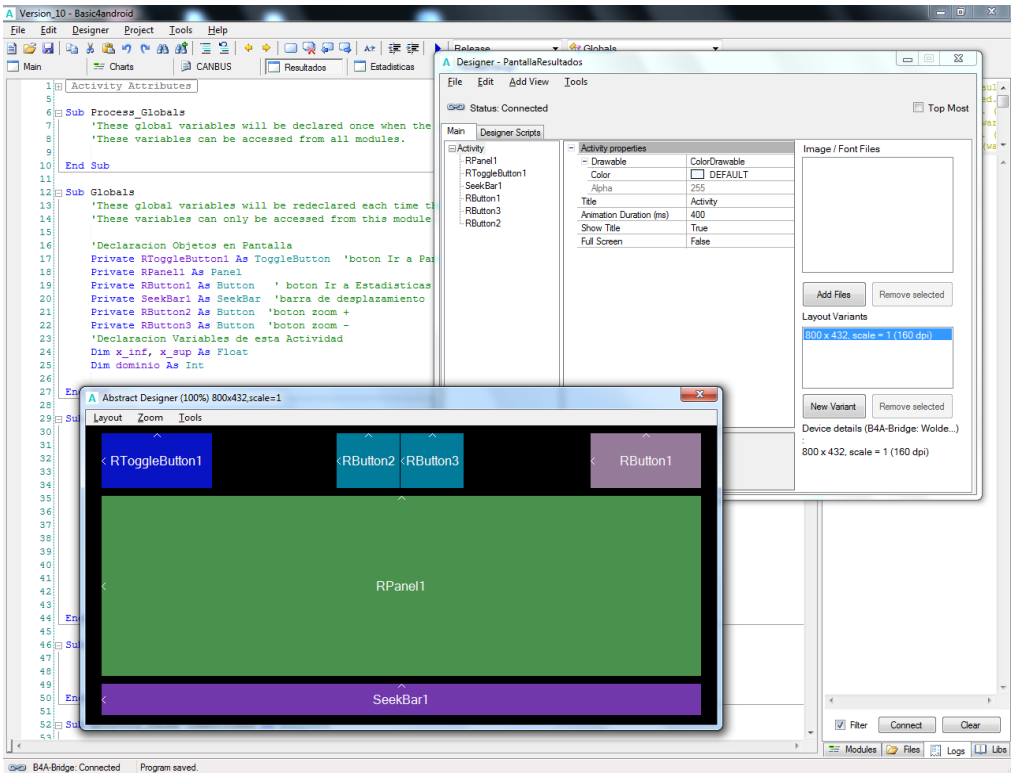

Figura 8. Entorno de desarrollo B4A. Captura de pantalla.

7.2. Funcionalidad para el usuario y diseño de la interfaz gráfica (GUI)

Se ha optado por una tableta de 9 pulgadas (Modelo Wolder MiTab Chicago, véase Tabla 4), que permite un compromiso entre una visualización adecuada y su portabilidad. En este sentido la aplicación se ha desarrollado para una pantalla de este tamaño, diseñando todos los elementos de una manera que sean claramente visibles para el usuario:

Procesador	Cortex A7 – 2 x 1.0 GHz
Memoria RAM	512 MB
Almacenamiento	4 GB Flash NAND
Sistema Operativo	Android 4.4 KitKat

Tabla 4. Características de la tableta utilizada (Wolder MiTab Chicago)

El usuario tiene como opciones iniciar/detener la grabación, seleccionar el archivo de destino y mostrar los resultados de la evaluación del ciclo de conducción, tal y como se observa en la Figura 9:

Figura 9. Pantalla Inicio de la Aplicación. Fuente propia.

En la misma pantalla se dispone de un panel central que informa de la velocidad y la aceleración (en letra más pequeña) instantáneas. Finalmente, también se muestra el identificador del nodo y el tiempo transcurrido.

7.3. Estructura y funcionamiento del software desarrollado

7.3.1. Control de la tarjeta USBtin

Como se ha explicado anteriormente, la tarjeta se puede controlar enviando comandos en forma de cadenas de caracteres ASCII por el puerto USB. B4A cuenta con una librería que permite el control de puerto USB. Estas cadenas están definidas en los manuales de la tarjeta USBtin. Por ejemplo, para enviar un mensaje CAN, transmitimos por el puerto USB el comando siguiente (Tabla 5):

tiiildd..[CR]	Transmit standard (11 bit) frame. Iii: Identifier in hexadecimal format (000-7FF) l: Data length (0-8) dd: Data byte value in hexadecimal format (00-FF)

Tabla 5. Comando para la transmisión de un mensaje CAN

Es decir si enviamos t001411223344[CR], estamos enviando un mensaje CAN en el que:

- id=001h
- dlc=4
- data=11 22 33 44

Todos los comandos funcionan de manera similar tanto para enviar como para recibir mensajes. Una lista exhaustiva de los comandos se incluye como anexo.

7.3.2. Biblioteca USBCAN

Los comandos introducidos anteriormente permiten el control de la tarjeta, pero no son demasiado prácticos. Para facilitar su utilización en un proyecto anterior [1], éstos se integraron en una librería de B4A, que gestionaba la escritura de estos comandos junto con el puerto USB. Esta librería ha sido adoptada en su totalidad, pero revisada y comentada de nuevo. Así pues se cuenta con una librería que permite trabajar fácilmente con la tarjeta USBtin a partir de una serie de funciones integradas en la librería.

Así pues, enviar un mensaje por el bus CAN se haría mediante la llamada a la función *canWrite*:

```
canWrite(id As Int, msg() As Byte, dlc As Int, mask As String)
```

7.3.3. Gráficos

B4A posee la funcionalidad de representar gráficos. Se han utilizado las funciones para representar la gráfica de la velocidad vs tiempo de manera dinámica (posibilidad de hacer zoom), tal y como muestra la Figura 10Figura 9.

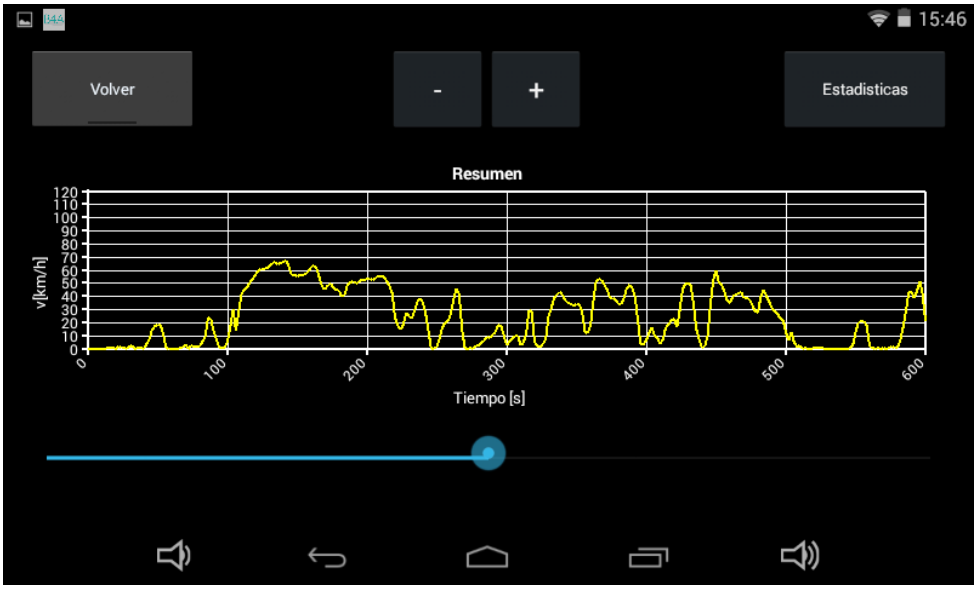

Figura 10. Representación de la velocidad en el ciclo de conducción.

7.3.4. Caracterización del ciclo de conducción

Se ha implementado una rutina que realiza las funciones explicadas en el capítulo anterior relativas a la caracterización del ciclo de conducción, devolviendo como salida las figuras de mérito más relevantes. En la Figura 11 se muestra un resumen de las estadísticas del ciclo de conducción que devuelve la aplicación.

Figura 11. Estadísticas del Ciclo de Conducción

También se ha implementado la visualización gráfica de los resultados, tal y como se muestra en la *Figura 12*.

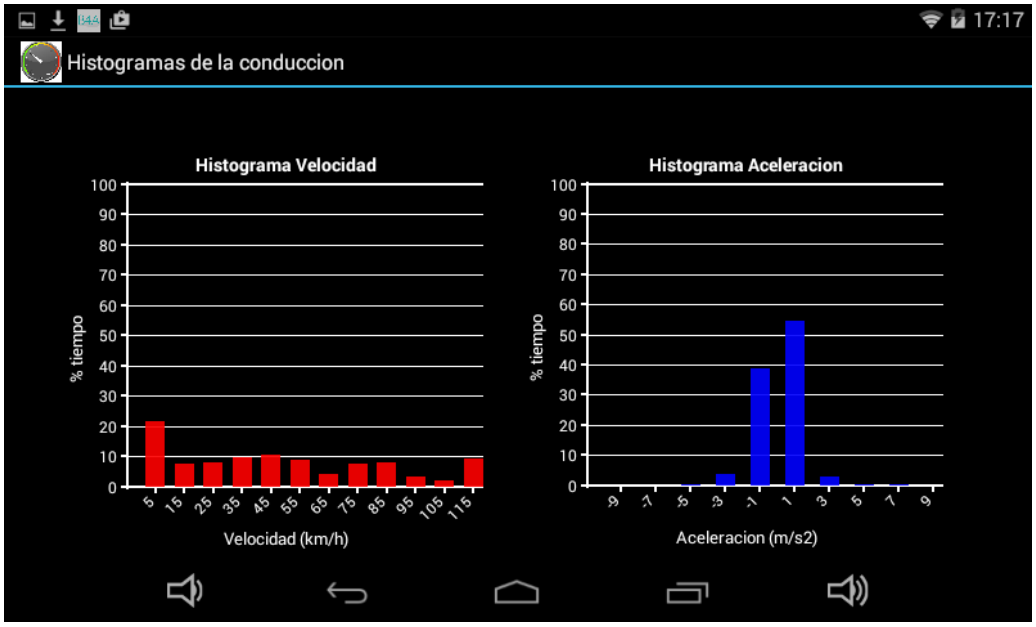

Figura 12. Visualización Gráfica Resultados. Elaboración propia.

Aquí se representan los histogramas de la velocidad (en rojo) y de la aceleración (en azul).

7.3.5. Programa Principal

El programa principal define las variables necesarias y gestiona las funciones vistas anteriormente, más las tareas de escritura a fichero del ciclo de conducción. Evidentemente, para recuperar los valores leídos de la velocidad se usa el estándar J1939, explicado en el capítulo 4.

El programa principal define el funcionamiento de los botones que se le presentan al usuario en la primera pantalla de la aplicación (véase Figura 9). Cuando el usuario inicia la aplicación se han de realizar las siguientes tareas:

1) Inicializar la conexión USB (con velocidad de transmisión 115200 bps emulando a un puerto serie)

2) Inicializar el bus CAN (250 Kbps)

3) Inicializar el *timestamping*

4) Inicializar variables

5) Definir el archivo de datos para guardar el registro de las velocidades del culo de conducción.

Que la velocidad de la conexión USB sea más baja que la velocidad del bus CAN no es problema porque nuestra aplicación sólo lee (y transmite) los datos de la velocidad de entre los muchos que pueden circular por el bus CAN. No obstante en un entorno real los datos no requeridos se han de filtrar mediante hardware.

Luego cuando el usuario decide que quiere iniciar la grabación del ciclo de conducción se inicializa el *timer* interno de la aplicación dando lugar al siguiente proceso que se repite cada 50 milisegundos (véase Figura 13)

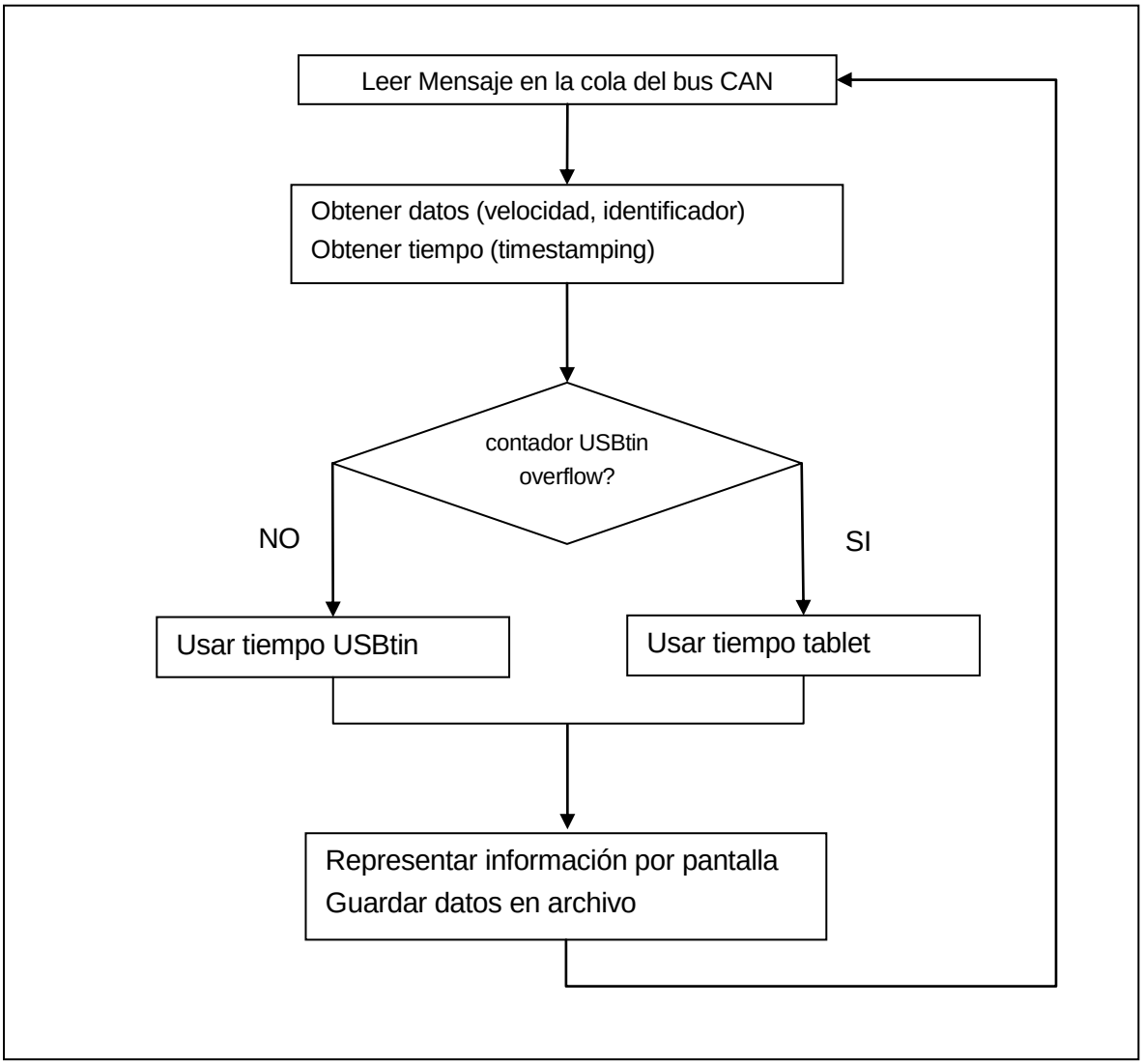

Figura 13. Proceso de lectura de los datos de la cola del bus CAN

En el proceso anterior se usa el contador interno (*timer*) de la tarjeta USBtin siempre que sea posible. No obstante, como máximo puede contar hasta unos 6 segundos, tras lo cual se reinicia. Se ha de controlar si sucede esto, y en este caso utilizar el contador interno de la tableta.

Finalmente una vez el usuario finaliza la grabación del ciclo de conducción tiene la posibilidad de pasar nuevas pantallas controladas cada una de ellas controlada por los siguientes subprogramas:

7.3.6. Subprograma Resultados

Es el encargado de representar la gráfica de la velocidad en un intervalo de tiempo seleccionado (véase Figura 10). Para ello se han de buscar los puntos dentro de la selección y remuestrearlos para adaptarlos a la resolución en pantalla. Se hace uso de la librería Charts para la representación de la gráfica.

7.3.7. Subprograma Estadísticas

Aquí (véase Figura 11) se realizan los cálculos de diversos parámetros del ciclo de conducción (tiempo transcurrido, velocidad media, distancia recorrida, aceleración, desviación de la velocidad, tiempo detenido, aceleraciones y frenadas bruscas, y del indicador de la agresividad.

7.3.8. Subprograma Histogramas

Se representan los histogramas de la velocidad y de la aceleración (véase Figura 12).

8. Test y Validación

En este capítulo se describe el proceso de validación de la aplicación desarrollada. En primer lugar se comprueba que la aplicación queda instalada correctamente en el menú de la tableta (véase icono *CicloConduccion* en la Figura 14).

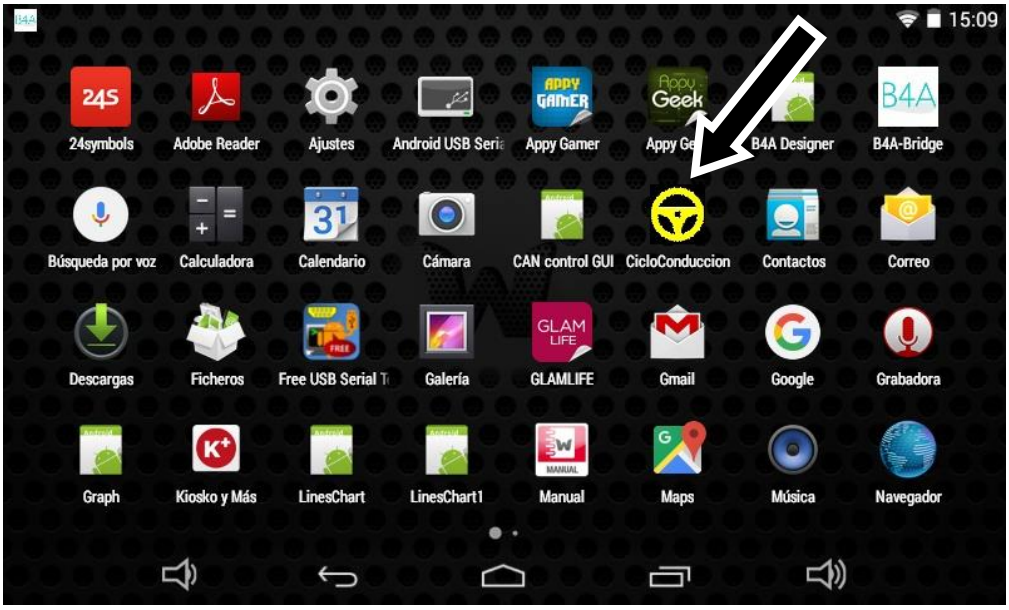

Figura 14. Icono de la aplicación instalada en el menú de la tableta. Fuente: Elaboración propia.

8.1. Configuración utilizada

Puesto que no fue posible disponer de un camión para realizar pruebas conectándose a su bus CAN las pruebas de validación se realizaron emulando el funcionamiento del bus CAN de un camión tal como se explica a continuación.

Además de la tarjeta USBtin, el Laboratorio de Electrónica posee un conversor CAN/USB de Kvaser que permite generar mensajes CAN a través de un PC. Esto nos permite, por una parte, emular el tráfico de los mensajes CAN que se producen durante la conducción de un vehículo a partir de los datos guardados en un archivo de texto. Para ello se ha adaptado un programa escrito en Python en el PC (véase código fuente en el anexo).

Este programa lee un fichero de texto con los datos de las velocidades de un ciclo de conducción y lo envía por un bus CAN mediante mensajes periódicos (en el programa se ha de indicar la frecuencia de muestreo). En este programa también se define el identificador utilizado (en nuestro caso el correspondiente al grupo de parámetros *Cruise Control /Vehicle*

Speed: 18FEF100h) para construir los campos necesarios del bus CAN.

Para realizar el test sobre el sistema desarrollado se ha utilizado la siguiente configuración mostrada a continuación en la Figura 15.

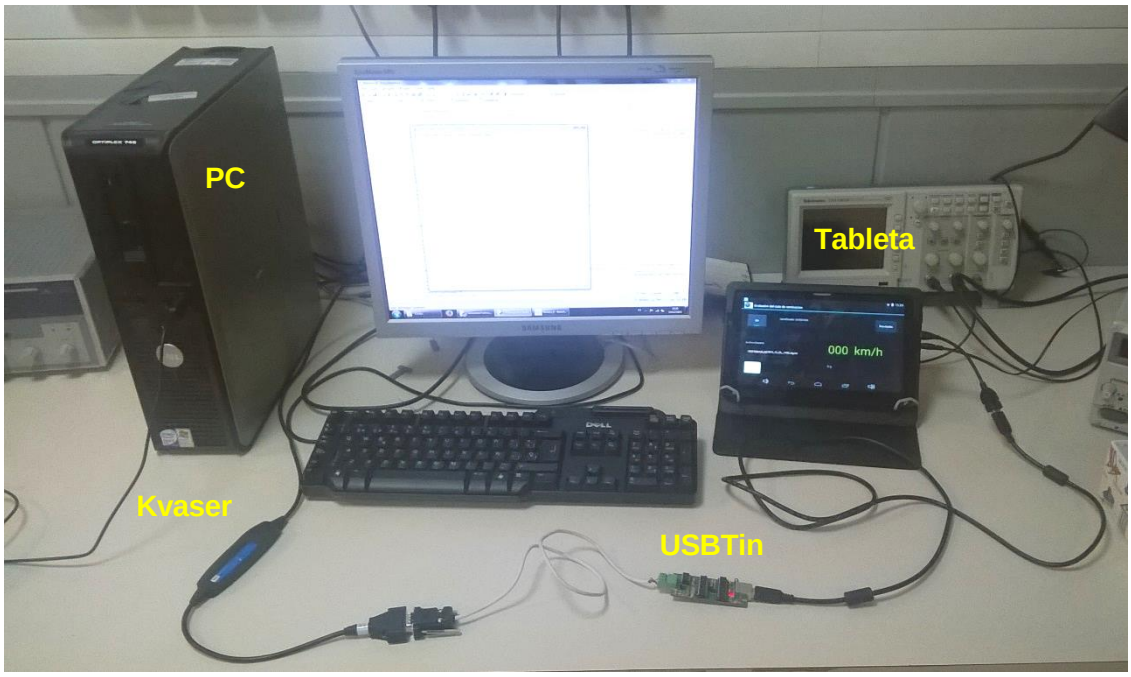

Figura 15. PC conectado a Kvaser, USBtin y tableta para el testeo de la aplicación.

Por otra parte, conectamos nuestro sistema compuesto de la tarjeta USBtin y la tableta al conector CAN del módulo Kvaser y comprobamos la recepción de la información enviada por el PC, que se registra en la tableta.

8.2. Ciclos de Conducción Sintetizados

Por tanto, para realizar las pruebas según lo explicado en el anterior apartado es necesario disponer de registros de la velocidad cada 100 ms en un archivo de texto. Para generar estos archivos podemos recurrir a ciclos de referencia (Ciclo Europa, etc.) [22] o bien sintetizarlos mediante un software [23] desarrollado por TNO (véase Figura 16).

No obstante el periodo de muestreo que se proporciona como resultado es de 1 segundo, mientras que el bus CAN transmite cada 100 milisegundos, por lo que se hizo necesario remuestrear la señal, lo cual se realiza fácilmente en Matlab. Por otra parte también se puede aumentar la longitud de un ciclo concatenando ciclos de menor duración.

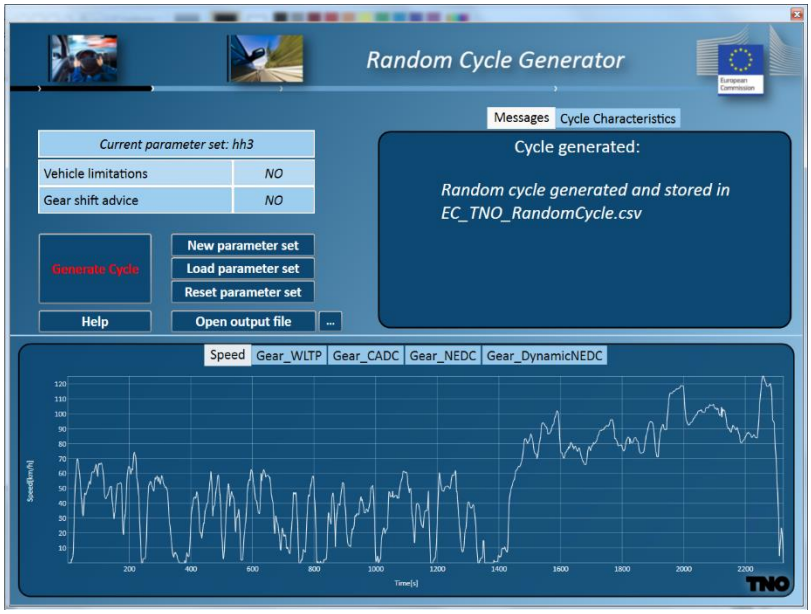

Figura 16. Captura de pantalla Random Cycle Generator [23].

8.3. Resultados

Después de diferentes pruebas con diferentes ciclos de conducción se comprueba que el sistema es capaz de leer correctamente el ciclo de conducción generado. Por otra parte, el sistema realiza la caracterización del ciclo de conducción adecuadamente.

Además tal y como se muestra en la siguiente Figura 17, la aplicación genera un archivo de texto con el registro de los datos de la velocidad transmitidos por el bus CAN.

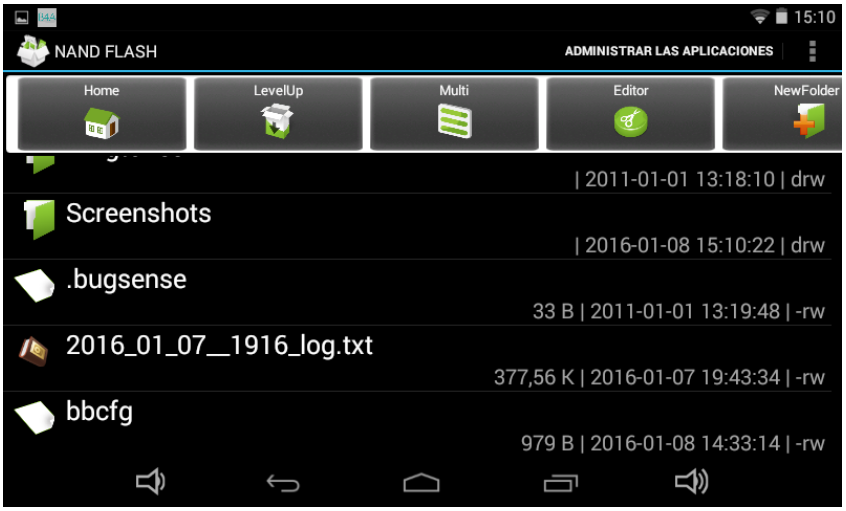

Figura 17. Archivo de texto (2016_01_07_1916.txt) generado por la aplicación. Fuente: Elaboración propia.

El anterior archivo de texto se genera en tiempo real y permite recuperar los datos si el sistema sufre una avería (fallo de la alimentación, etc.)

En el archivo de texto (Figura 18) queda registrado lo siguiente: tiempo interno de la tableta, tiempo interno de USBtin, velocidad (km/h) e intervalo entre mensajes (segundos).

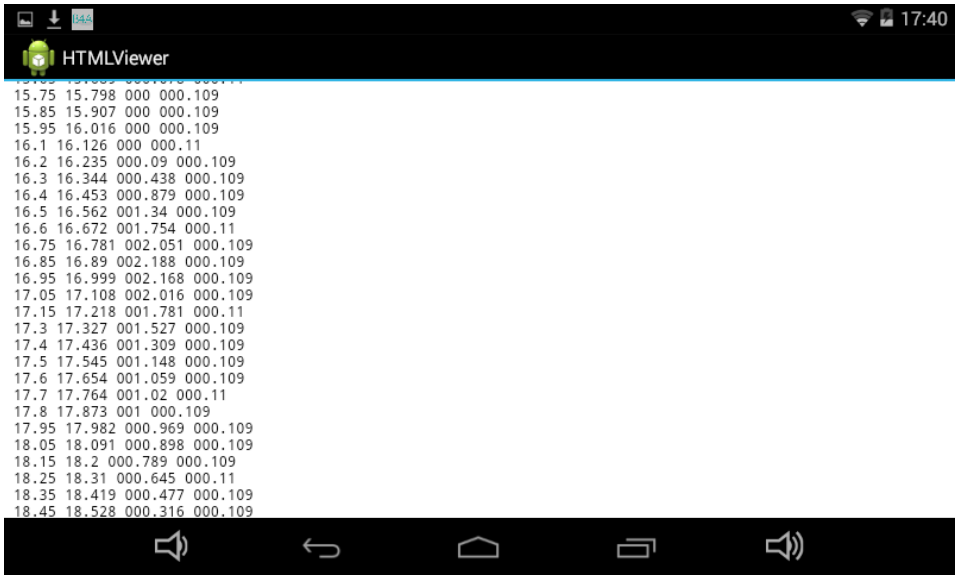

Figura 18. Vista del archivo de texto generado.

Una vez se ha determinado que la aplicación se ejecuta correctamente se procede a comprobar que la definición de agresividad que se ha adoptado tiene sentido, y que el indicador de agresividad adoptado proporciona medidas más bajas en un ciclo con una conducción suave (poco agresiva), y medidas más elevadas en un ciclo de conducción más agresivo.

8.3.1. Caso A: ciclo de conducción suave

El Ciclo Europa es una prueba diseñada para evaluar objetivamente el impacto medioambiental de los automóviles. La prueba reproduce un patrón de aceleraciones suaves, velocidades de crucero constantes y muchos períodos al ralentí por lo tanto es de esperar una agresividad baja. Los resultados obtenidos en nuestra aplicación se representan en las Figuras 19, 20 y 21.

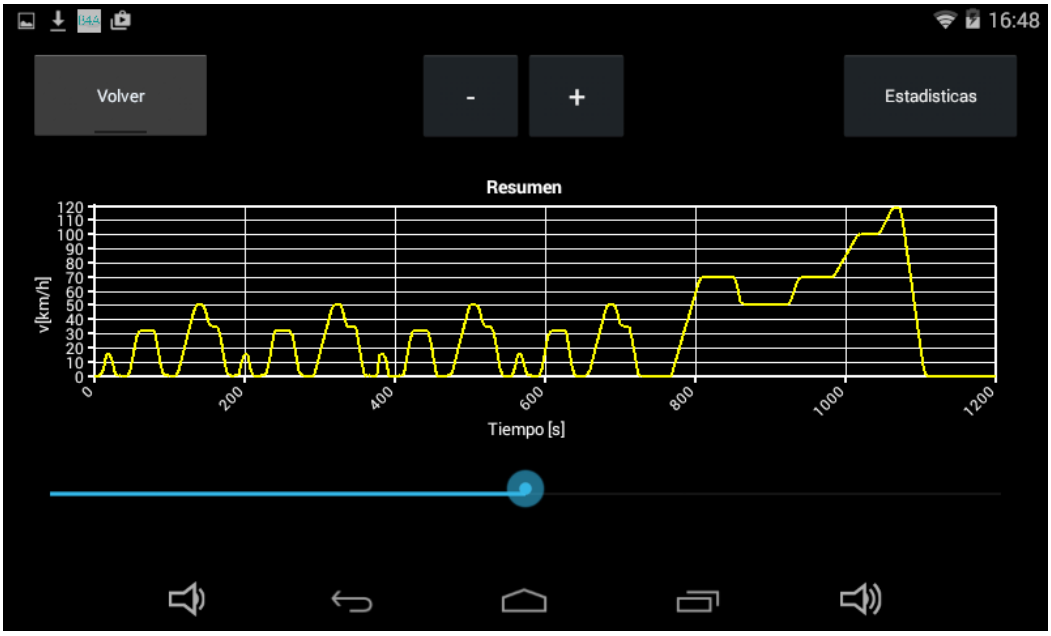

Figura 19. Gráfica v(t). Fuente: Elaboración propia.

Figura 20. Estadísticas caso A. Fuente: Elaboración propia.

Por tanto en este caso el índice de agresividad obtenido es de 12. Es un ciclo muy suave, sin frenadas ni aceleraciones fuertes.

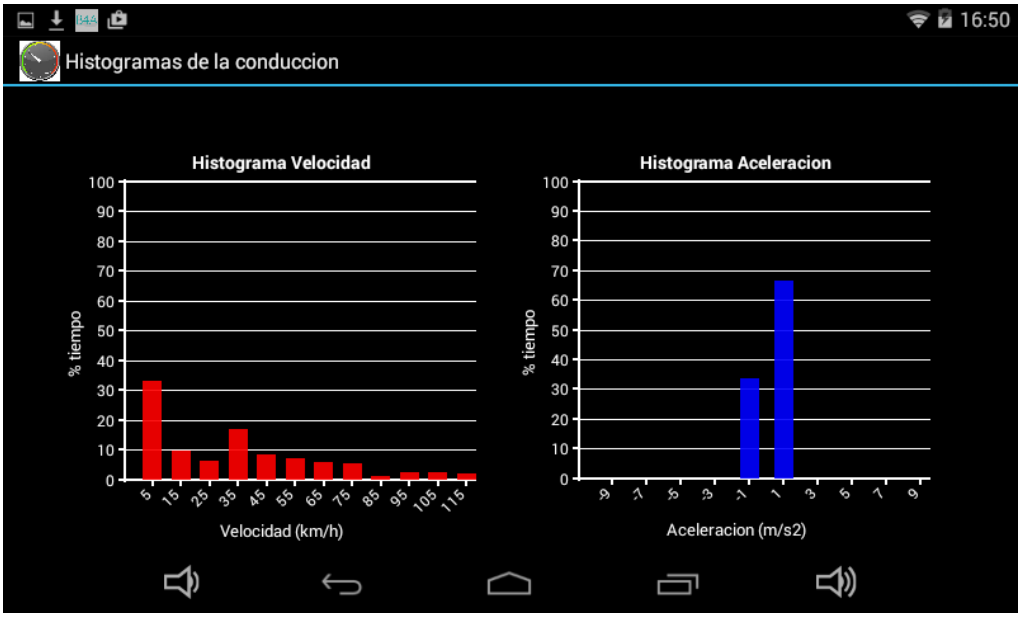

Figura 21. Histogramas caso A . Fuente: Elaboración Propia.

8.3.2. Caso B: ciclo de conducción agresivo

En este caso imponemos un ciclo con mucha más variación de las velocidades (Figuras 23, 24 y 25)

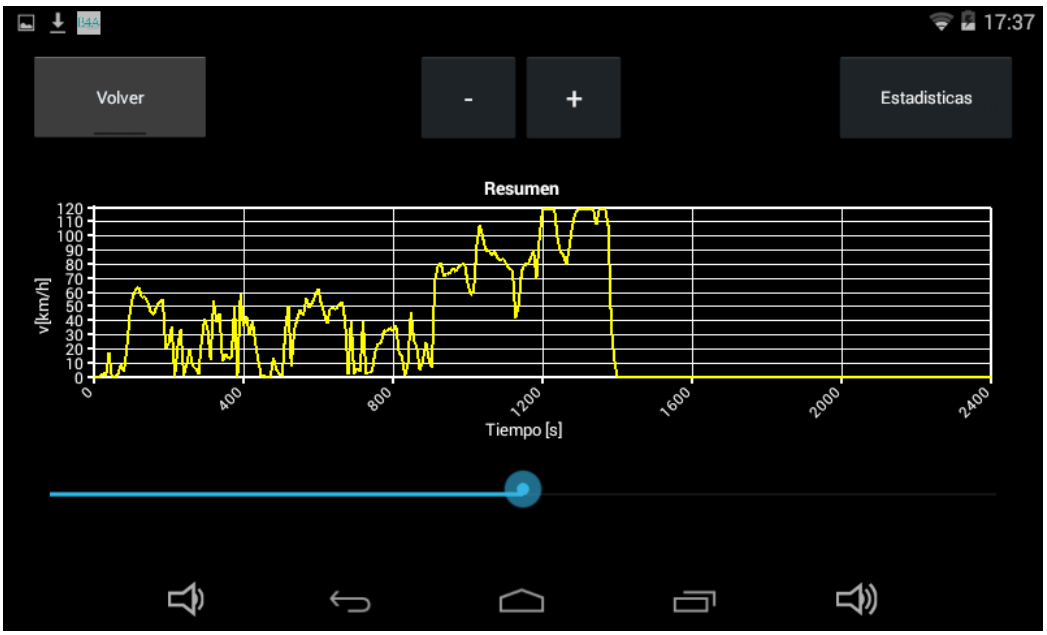

Figura 22. Gráfica v(t). Fuente: Elaboración propia

Figura 23. Estadísticas caso B. Fuente: Elaboración propia.

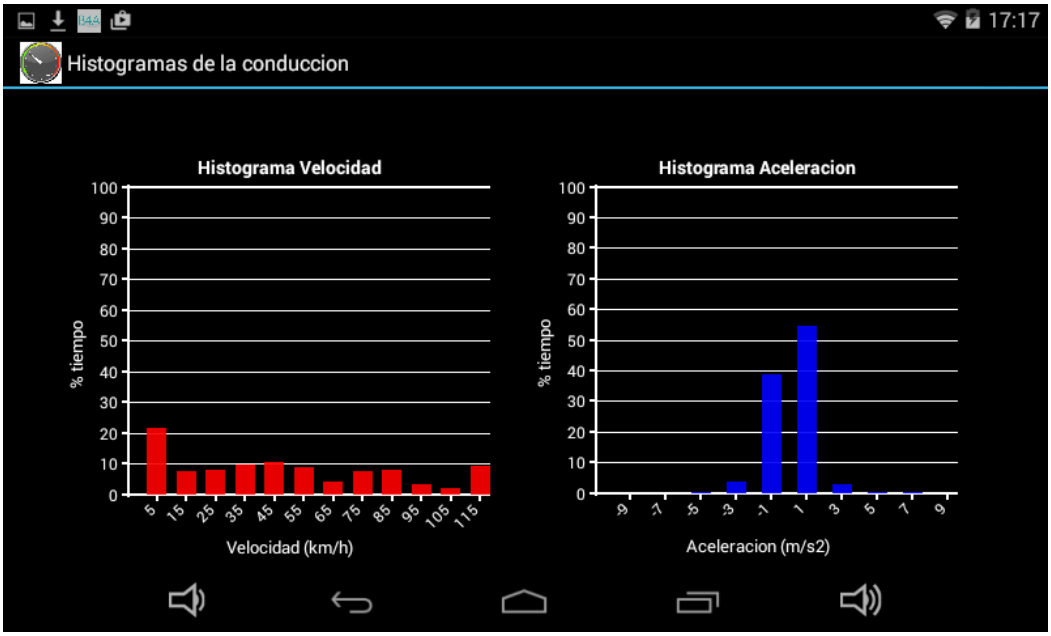

Figura 24. Histogramas caso B . Fuente: Elaboración Propia.

En este caso la agresividad obtenida es de 131, claramente mayor que en el caso anterior.

8.3.3. Comparación agresividad entre ciclos

Si comparamos la agresividad obtenida anteriormente observamos que efectivamente el indicador de agresividad utilizado se comporta de la manera esperada, incrementándose con la agresividad del ciclo, que es el comportamiento deseado.

9. Impacto Ambiental

Con respecto al impacto ambiental de este proyecto solo cabe decir que no es necesario un análisis detallado debido al reducido impacto contaminante que tendrá. En todo caso, atendiendo al ciclo de vida completo del proyecto, se podría considerar lo siguiente:

En la fase de desarrollo del proyecto se generará un pequeño impacto ambiental causado, por una parte, por la adquisición de los componentes (tableta y tarjeta interfaz), dado que se han necesitado materiales y energía para su fabricación y comercialización. También se puede considerar un pequeño consumo de energía eléctrica en el desarrollo de software.

En la fase de funcionamiento del proyecto se espera que este tenga un impacto positivo al mejorar la conducción de los vehículos, suponiendo cierto ahorro en el consumo de energía y una disminución de los contaminantes. El resultado de este proyecto permite implementar mecanismos para premiar actitudes ecológicas en la forma de conducir y penalizar modos de conducción poco responsables con el medio ambiente.

En la fase de desmantelamiento se tendría que considerar el impacto de los residuos generados por los componentes electrónicos. Esto implica un gasto de energía adicional y una generación de residuos, si bien se pueden recuperar ciertos materiales.

10. Planificación Temporal

La planificación temporal se muestra en la figura siguiente (Figura 25):

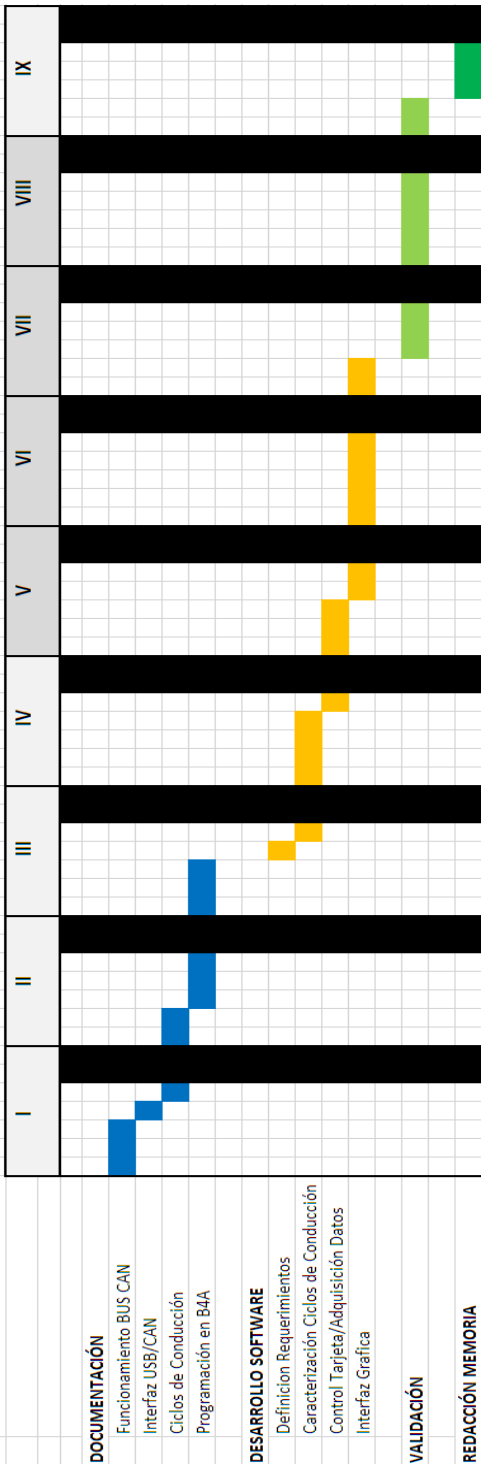

Figura 25. Planificación Temporal.

11. Presupuesto

El presupuesto estimado para la realización de este proyecto es el siguiente (Tabla 6):

Unidades		**Precio Unitario (€/ud.)**		**Total (€)**
1	Tableta	60		60
1	USBtin	30		30
1	Licencia B4A	50		50
100	Documentación	50		5000
150	Desarrollo (horas)	50		7500
75	Test y Validación (horas)	50		3750
25	Redacción memoria(horas)	50		1250
			TOTAL	**17640**

Tabla 6. Presupuesto.

El presupuesto de desarrollo del proyecto se establece en 17.640 Euros (DIECISIETE MIL SEISCIENTOS CUARENTA EUROS).

El autor del proyecto

Conclusiones

En este trabajo se ha desarrollado un sistema de bajo coste basado en el uso de una tableta Android que a través de una conexión a un bus CAN permite obtener información sobre el grado de agresividad de un ciclo de conducción.

Debido a las restricciones impuestas por la industria de automoción el ámbito de aplicación queda restringido a camiones y autobuses, donde el protocolo de alto nivel que es necesario para interpretar la información que se obtiene del bus está estandarizado (SAE J1939) y es de dominio público.

Con este objetivo se ha realizado un trabajo de documentación sobre el bus CAN, así como sobre el estándar J1939 y los códigos asociados a este estándar.

También ha sido necesario programar una aplicación en B4A para el control de la tarjeta USBtin, y el procesado de la información recibida. Se ha implementado, además, la caracterización del ciclo de conducción y la visualización de los resultados obtenidos.

Por último, se ha comprobado que la aplicación funciona de la manera prevista, permitiendo el registro y análisis del ciclo de conducción.

Recomendaciones y Mejoras futuras

Puesto que el uso del bus CAN proporciona una información limitada se recomienda o bien conseguir información del fabricante sobre la codificación de la información o complementar los datos con un sistema externo (acelerómetros, GPS).

En este sentido, si se dispone de una mayor cantidad de parámetros, se puede mejorar la caracterización del ciclo de conducción.

Otra mejora en el sistema seria disponer de un interfaz inalámbrico que aumente la comodidad de uso del sistema.

Finalmente sería interesante poder testear el sistema en un vehículo real en lugar de una emulación.

Agradecimientos

El autor quiere expresar su agradecimiento a su director, Manuel Moreno, por su guía a lo largo de la realización de este Trabajo Final de Máster, así como por su disponibilidad y amabilidad en todo momento.

Bibliografía

Referencias bibliográficas

[1] M. Garrido, “Interfaz Gráfica de usuario usando una tableta Android y un adaptador CAN/USB,” UPC, 2015.

[2] T. Fischl, “USBtin - USB to CAN interface,” 2015. [Online]. Available: http://www.fischl.de/usbtin/.

[3] SAE, “SAE J1939 Standards Collection.” [Online]. Available: SAE J1939 Standards Collection. [Accessed: 01-Oct-2016].

[4] Wikipedia, “CAN bus.” 2015.

[5] M. Di Natale, “Understanding and using the Controller Area Network,” 2008.

[6] S. Corrigan, “Introduction to the Controller Area Network (CAN),” 2008.

[7] Robert Bosch GmbH, “CAN Specification Version 2.0,” 1991.

[8] Cowfish Studios, “OBD-Pi: Raspberry Pi Displaying Car Diagnostics (OBD-II) Data On An Aftermarket Head Unit,” 2015. [Online]. Available: http://www.cowfishstudios.com/blog/obd-pi-raspberry-pi-displaying-car-diagnostics-obd-ii-data-on-an-aftermarket-head-unit.

[9] SAE, “Vehicle Application Layer.” [Online]. Available: http://standards.sae.org/j1939/71_201309/. [Accessed: 01-Oct-2016].

[10] Vector Informatik GmbH, “Networking Heavy-Duty Vehicles Based on SAE J1939,” 2008.

[11] Schneider-Electric, “SAE j1939 spreadsheet.” [Online]. Available: http://www2.schneider-electric.com/library/SCHNEIDER_ELECTRIC/SE_LOCAL/APS/205363_1726/sae_j_1939_spreadsheet_supported_by_TwidoExtreme.xls.

[12] Anonymous Contributors, “CAN Interface Collection,” *CAN Wiki*, 2015. [Online]. Available: http://www.can-wiki.info/doku.php?id=can_interfaces:main.

[13] Kvaser, “Kvaser Leaf Light HS,” 2014. [Online]. Available: https://www.kvaser.com/products/kvaser-leaf-light-hs/#/! [Accessed: 10-Jan-2016].

[14] Z. Dai, D. Niemeier, and D. Eisinger, “Driving cycles: a new cycle-building method that better represents real-world emissions,” *U.C. Davis-Caltrans Air Qual. Proj.*, vol. 66, no. 66, p. 37, 2008.

[15] M. P. O. Keefe, A. Simpson, K. J. Kelly, and D. S. Pedersen, “Duty Cycle Characterization and Evaluation Towards Heavy Hybrid Vehicle Applications,” *SAE*

World Congr. Exhib., no. April, p. 12, 2007.

[16] A. Duran and K. Walkowicz, “A Statistical Characterization of School Bus Drive Cycles Collected via Onboard Logging Systems,” *SAE Int. 2013-01-2400*, pp. 400–406, 2013.

[17] A. Belén, R. González, M. R. Wilby, J. José, V. Díaz, and C. S. Ávila, “Modeling and Detecting Aggressiveness From Driving Signals,” vol. 15, no. 4, pp. 1419–1428, 2014.

[18] A. Bolovinou, A. Amditis, F. Bellotti, and M. Tarkiainen, “Driving Style Recognition for Co-operative Driving: A Survey,” *Adapt. 2014, Sixth Int. Conf. Adapt. Self-Adaptive Syst. Appl.*, no. c, pp. 73–78, 2014.

[19] L. L. Hoberock, “A Survey of Longitudinal Acceleration Comfort Studies in Ground Transportation Vehicles,” *J. Dyn. Syst. Meas. Control*, vol. 99, no. 2, p. 76, 1977.

[20] E. K. Nam, C. a Gierczak, and J. W. Butler, “A Comparison Of Real-World and Modeled Emissions Under Conditions of Variable Driver Aggressiveness,” *TRB 2003 Annu. Meet.*, no. 313, 2003.

[21] Anywhere Software, “B4A - The simplest way to develop real-world, native Android apps,” 2015. [Online]. Available: https://www.b4x.com/.

[22] T. Barlow, S. Latham, I. Mccrae, and P. Boulter, “A reference book of driving cycles for use in the measurement of road vehicle emissions,” 2009.

[23] TNO, “Random Cycle Generator.” [Online]. Available: https://www.tno.nl/en/focus-area/urbanisation/mobility-logistics/clean-mobility/random-cycle-generator/.